UNIVERSITÀ DEGLI STUDI DI UDINE
HIC SUNT FUTURA

DEPARTMENT OF MATHEMATICS, COMPUTER SCIENCE AND PHYSICS

MASTER THESIS IN
INFORMATICA

# Translating unitaries into the Measurement Based Quantum Computing model

CANDIDATE
Diego Borsoi

SUPERVISOR
Prof. Carla Piazza

CO-SUPERVISOR
Dott. Riccardo Romanello

Academic Year 2022/2023

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

# Acknowledgements

First of all, a big thanks goes to Prof. Carla Piazza and Dott. Riccardo Romanello for their help in writing this thesis and for their patience with me, even with such a limited time constraint.

A thanks goes also to my family, who has supported me for all these years, even if this master's degree took longer than it should have.

All my friends need to be thanked too because they encouraged me to hurry up, even if their concern was primarily when we were going for a drink.

Last but not least, a special thanks goes to my wonderful girlfriend, Elena. She supported me and helped me stay sane, making my days easier and making me think about other things on my days off.

# Abstract

In the Measurement Based Quantum Computing model (MBQC), and in particular in the One-Way Quantum Computing model (1WQC) [14], the computation is performed measuring qubits prepared in an highly entangled state. For each qubit one has to determine the bases in which it is measured. Moreover, this measurements need to be executed in a specific order, ensuring a deterministic computation.

In the years after the proposition of this new technique the effort has been centered on implementing known gates (already used in the circuit model). These gates apply unitary transformation to the qubits thus allowing the execution of quantum algorithms.

A technique to implement any unitary matrix directly in the 1WQC model, deriving measurements bases and order of execution, has been proposed in [17]. This thesis presents an in dept analysis of the algorithms presented giving some insights on the techniques that can be used for an implementation. The third part of the algorithm discussed in [17] has been replaced by the better version introduced in [25] for the calculation of the causal flow. At the end, an analysis on the space and time complexity of the algorithms is presented.

# Contents

# 1

# Introduction

Quantum mechanics places its origin in the early 90s with the introduction of quantized energy levels by Max Plack. In the following decades various scientists (among them Einstein, Bohr and Heisenberg to name a few) developed numerous theories evolving the field. In the same time this new theory was applied in various areas of research.

It was only between the 70s and 80s that the concept of Quantum Information Theory arose [1], applying the concept of Quantum Mechanics to the field of computer science. After this first approach various other scientists expanded the area. In 1980 Paul Benioff proposed the idea of Quantum Turing Machine [2]. This model provided a way to capture the power of the quantum world. Based on this concept, in 1982, Richard Feymann proposed the idea of a quantum simulator capable of imitating the evolution and complexity of physics [3]. Finally, in 1985, David Deutsch introduced the concept of Quantum Computer [4] demonstrating the capability of this model. He proved that this type of computer could execute any computation that any classical one could do and also perform quantum computations, feasible in normal computers only with an at least exponential speed up.

This paper laid the foundation to the development of quantum algorithms. The first of this algorithms was the Deutsch–Jozsa algorithm [6]. This revolutionary paper demonstrated the first instance in which a quantum computer could out-speed a normal one. After this breakthrough more algorithms where discovered, expanding the list of cases in which the quantum computer outperforms the classical one.

With this first quantum algorithms been created, there was the need to represent the quantum computation. This need gave life to the *circuit model*. This description of a quantum computation takes inspiration from the classical circuitry. This model consists of:

- one line for each qubit used,
- gates applied to 1 or more qubits,
- measurements on one qubit.

Intuitively the computation takes place from left to right, applying in order each gate/measurement to the set of qubits. With the creation of this technique were also proposed various gates implementing numerous operations.

A limitation of this model is that the overall transformation need to be reversible and unitary. In other words the computation can be represented as a unitary matrix. This matrix is applied to the initial qubits giving the result of the computation. The problem is thus implementing this type of matrix using simple gates. Techniques were discovered to decompose unitary matrices ([26] chapter 4.5.1), and also in the present days lots of effort is spent in optimizing the number of base gates generated [30][33][34] or reducing particular types of gates [31][35].

In 2001 a new model for performing quantum computation has been presented by Raussendorf, Browne and Briegel giving birth to the *Measurement Based Quantum Computation* (MBQC) [11], later refined in a paper published in 2003 [14]. In particular they presented a version of MBQC called *One-Way Quantum Computing* (1WQC).

This particular model works in a totally different way than the circuit one. The 1WQC consists of only 2 elements:

- a graph state of qubits,
- measurements on the qubits.

The algorithm is executed starting from a graph state of qubit in an highly entangled state. Then at each step some of this qubits are measured advancing the computation. This measurements are executed changing the bases in which they are performed, exploiting the connection between qubits to progress the computation.

After the proposal of the 1WQC many researchers have focused their research on this new model. All the gates used in the circuit model has been translated to this new one [14]. Some of the most famous algorithms have also been implemented in this new model [28][29].

In this new model of computation persist the problem presented for the circuit one: the implementation of unitary matrix. A simple approach could be to use the techniques found for the circuit model and break down the matrix in simpler gates, implementing this using the known techniques. But a question arises: *can a unitary matrix be implemented directly, i.e. without passing through the decomposition in simpler gates?*

An answer to this problem has been proposed by de Beaudrap, Danos and Kashefi in 2006 [17]. But to this day the technique has not been implemented or further developed. Only the last part of the algorithms presented in that paper has been refined. The causal flow problem has been improved numerous times [19][22][23][24], reaching the last version in [25].

This thesis aims at a more in-depth analysis of the algorithms presented in [17] discussing useful details for an implementation. Is also presented a study on the complexity of these techniques, giving a space and time estimation of the resources needed for the creation of the 1WQC algorithm. This research is structured as follows.

In the next chapter 2 are provided the preliminary requirements. First of all a recall of some notions of linear algebra and graph theory. After that are presented the principles at the base of Quantum Computation. Followed by a description of the circuit model and the MBQC, with particular emphasis on the 1WQC.

The chapter 3 illustrates the algorithm of [17] and [25]. Giving a brief description of the steps.

After that, in chapter 4, an analysis of the space and time complexity of the algorithms is presented, defining also possible techniques for the implementation.

In the end, conclusions and potential further developments are considered in chapter 5.

# 2

# Preliminaries

This chapter presents some basic notions needed for the understanding of the algorithms that will be presented.

First of all, a recall is given on some basic notions of complex numbers, linear algebra and graph theory with the focus on specifying the notation used. After that a description of the concepts at the foundations of quantum computing is presented, followed by a description of the two main models: circuit model and MBQC, with a particular focus on the 1WQC paradigm.

For an in-depth view of these basic concepts, the reader is advised to consult the book by Nielsen and Chuang on the matter [26].

## 2.1 Complex numbers

At the base of all the computations that will be presented there are *complex numbers*. These numbers are composed by adding a real number with an imaginary one. They thus have the form:

$$z = a + bi$$

with $a$ and $b$ real numbers and the indeterminate $i$ satisfying the equation $i^2 = -1$. The numbers $a$ and $b$ are called the *real* and *imaginary* part, respectively. In other words, a complex number represents a point in the complex plane: a Cartesian plane in which the $x$-axis is formed by real numbers (real axis) and the $y$-axis by imaginary ones (imaginary axis). As a result of this construction, simple operations (addition, subtraction and multiplication) can be done in the intuitive way.

The *complex conjugate* $z^*$ of a complex number is defined as a complex number with the same real part and the imaginary part equal in magnitude but opposite in sign. In other words, this transformation changes the sign of $i$.

$$z^* = a - bi$$

Another operation that will be used must be defined. The *norm* of a complex number:

$$|z| = \sqrt{a^2 + b^2}$$

A complex number can be expressed in an alternative form. The *polar form* represents the distance $r$ and angle $\alpha$ (with respect to the real axis) of the segment that connects the $z$ point with the origin of the axis.

$$z = r(\cos\alpha + i\sin\alpha)$$

Using *Euler's formula* ($e^i\alpha = \cos\alpha + i\sin\alpha$) a complex number can be represented in a further way:

$$z = re^{i\alpha}$$

If the complex number has a norm of 1, it can be represented in a simpler way (being that $r = 1$):

$$z = a + bi = \cos\alpha + i\sin\alpha = e^{i\alpha}$$

## 2.2 Linear Algebra

A column vector $v$ (usually called vector, dropping the column prefix) composed of $n$ elements has the form:

$$v = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

If the elements $a_1, ..., a_n$ are complex numbers, the vector represents an element in a complex vector space $\mathbb{C}^n$.

A vector can undergo a transformation called *transpose*. This operation creates a row vector of the form:

$$v^T = [a_1, a_2, ..., a_n]$$

On vectors of the complex space a slightly different transformation can be applied called *conjugate transpose*. This operation consists of the application of the transpose and the complex conjugate of each element. Practically:

$$v^\dagger = (v^*)^T = (v^T)^* = [a_1^*, a_2^*, ..., a_n^*]$$

In the quantum computation field a notation is used called *Bra-ket notation*, or *Dirac notation*. It introduces two elements:

- **bra** written as $|v\rangle$. Denotes a vector $v$ in a complex vector space.
- **ket** written as $\langle f|$. Denotes a linear map $f : \mathbb{C}^n \to \mathbb{C}$ mapping each vector in $\mathbb{C}^n$ to a complex number. A ket can also be viewed as a row vector.

This notation is used to ease the types of calculations that frequently come up.

In particular, this notation is useful because the vectors used belongs to a finite dimension *Hilbert space* $\mathcal{H}$. In our case, this is a complex space with an *inner product operator*. This operator is a map of the form $(\cdot,\cdot) : \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$ and needs to satisfy the following properties. For all vectors $x, y, z \in \mathbb{C}^n$ and scalars $a, b \in \mathbb{C}$:

1. *conjugate symmetry*: $(x, y) = (y, x)^*$
2. *linearity in the first argument*: $(ax + by, z) = a(x, z) + b(y, z)$
3. *positive-definiteness*: $(x, x) \geq 0$, with equality only when $x$ is a vector of zeros

With the bra-ket notation one can define the inner product of the elements

$$\langle B| = [b_1, b_2, ..., b_n] \quad \text{and} \quad |A\rangle = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

as:

$$\langle B|A\rangle = [b_1, b_2, ..., b_n] \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = b_1a_1 + b_2a_2 + ... + b_na_n$$

This operation respects all the properties mentioned before. Based on this definition, one can transform a bra in a ket and vice versa using the conjugate transpose operator:

$$|v\rangle^\dagger = \langle v| \quad \text{and} \quad \langle w|^\dagger = |w\rangle$$

A *matrix* $A$ is an $m \times n$ structure ($m$ rows and $n$ columns) of the form:

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & ... & a_{mn} \end{bmatrix}$$

The multiplication of a matrix by a scalar is done intuitively by multiplying each element of the matrix by that value.

Given two matrices $A$ ($m_1 \times n_1$) and $B$ ($m_2 \times n_2$), with $n_1 = m_2$, the product is defined as:

$$A \times B = AB = \begin{bmatrix} c_{11} & c_{12} & ... & c_{1n_2} \\ c_{21} & c_{22} & ... & c_{2n_2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m_1 1} & c_{m_1 2} & ... & c_{m_1 n_2} \end{bmatrix} \quad \text{with} \quad c_{ij} = \sum_{k=1}^{n_1} a_{ik} b_{k,j}$$

A *square matrix* is a matrix with the same number of rows and columns ($m = n$). A square matrix with ones on the main diagonal and zeros everywhere else is called an *identity matrix* $\mathbb{I}$. Even for the matrices the operator of conjugate transpose is defined. This is applied by swapping the rows of a matrix with the columns of itself and applying the complex conjugate of each element.

In this thesis, the matrices that will be dealt with are of a particular kind. They are called *unitary*

*matrix* $U$. This matrices are square matrices with a constraint:

$$UU^\dagger = U^\dagger U = \mathbb{I}$$

In other words, when pre or post multiplied by their conjugate transpose the result is an identity matrix. Another type of matrices are the ones that are equal to their conjugate transpose: $U = U^\dagger$. These are called *hermitian matrices*.

An important operation that will be used multiple times is the *tensor product*. Given two vectors $|v\rangle$ and $|w\rangle$ of the Hilbert spaces of $\mathbb{C}^n$ and $\mathbb{C}^m$ respectively, the result is a vector $|v\rangle \otimes |w\rangle$ of the Hilbert space of $\mathbb{C}^{nm}$. This vector is constructed as:

$$|v\rangle \otimes |w\rangle = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \otimes \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} v_1 w_1 \\ \vdots \\ v_1 w_m \\ v_2 w_1 \\ \vdots \\ v_2 w_m \\ \vdots \\ v_n w_1 \\ \vdots \\ v_n w_m \end{bmatrix}$$

This operation will be represented equivalently as $|v\rangle \otimes |w\rangle$, $|v\rangle|w\rangle$ or $|vw\rangle$.

The *Kronecker product* extend this operation to the matrices:

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

## 2.3 Graph theory

A *graph* $G = (V, E)$ is a pair of sets:

- $V$ the set of vertices (also called nodes),
- $E \subseteq \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$ the set of edges connecting two different nodes.

Taken any two nodes $x, y \in V$ there is an edge that connects these two nodes if and only if $\{x, y\} \in E$ (an edge will be written also as $xy$). This definition corresponds precisely to that of *undirected graphs*. The notation $|V|$ and $|E|$ represents the number of nodes and edges, respectively.

A slightly different structure will be used in this thesis: a *geometry* $(G, I, O)$ consists of an undirected graph $G(V, E)$ together with two subsets of nodes $I, O \subseteq V$, called *input* and *output*, respectively. The notation $I^c$ is used to indicate the set of nodes complements of $I$ in $G$ and $O^c$ for the set of nodes complements of $O$ in $G$.

Another construct that will be used is defined as follows. A *causal flow* $(f, \preccurlyeq)$ for a geometry $(G(V, E), I, O)$ consists of a map $f : O^c \to I^c$ and a partial order $\preccurlyeq$ over $V$, such that for all $x \in O^c$:

1. $\{x, f(x)\} \in E$
2. $x \preccurlyeq f(x)$
3. for all $y$ s.t. $\{y, f(x)\} \in E$ and $y \neq x$, then $x \preccurlyeq y$

Figure 2.1 shows a geometry with a flow. The map $f$ is represented as arrows from $O^c$ to $I^c$. The associated partial order is given by the labeled sets of vertices. The coarsest order $\preccurlyeq$ for which $(f, \preccurlyeq)$ is a flow will be called the *dependency order* induced by $f$. The number of partition sets (5 in the figure) is called the *depth* of the flow.

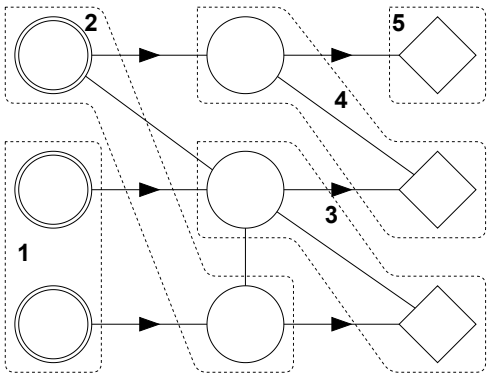



Figure 2.1: A geometry with causal flow. The double-circled nodes are the inputs $I$ and the diamonds are the outputs $O$. The flow map $f$ is represented by the arrows and the partial order on the vertices is given by the labelling of the 5 partition sets.

## 2.4 Quantum computing

A quantum computation gets its incredible abilities from the underlying implementation which exploits particular quantum mechanics properties. This thesis won't go into implementation details. The computation is viewed as a high theoretical level sequence of actions manipulating the quantum system. This system must adhere to some fundamental postulates. They are the following:

**1** *The state of an isolated physical system is represented, at fixed time t, by a state vector* $|\psi\rangle$ *belonging to a Hilbert space* $\mathcal{H}$ *called the* state space.

Let's take the simplest quantum system that will be used in this paper: the *qubit*. This is the base element of every quantum computation. The name derives from the classical counterpart, the *bit*, translated to the quantum world. Like the classical version, a qubit can take the 0 or 1 state. However, the fascinating part is that this element can be in every linear combination of these two states. In particular, the state of a qubit represents a vector in the state space $\mathbb{C}^2$.

Assuming the states 0 and 1 as represented by the computational bases $|0\rangle$ and $|1\rangle$, one can express the state of a qubit $|\psi\rangle$ as:

$$|\psi\rangle = a|0\rangle + b|1\rangle$$

where $a$ and $b$ are complex numbers. The qubit can thus be in a superposition state of the computational bases.

This postulate also imposes the vector $|\psi\rangle$ to be a *unit vector* (vector of length 1). From this is derived that:

$$|a|^2 + |b|^2 = 1$$

Being of unit length, a qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ (taken w.l.o.g. that $a$ is real and $b$ is complex) can be represented using the *Hopf coordinates* as:

$$a = \cos\left(\frac{\theta}{2}\right) \qquad b = e^{i\varphi}\sin\left(\frac{\theta}{2}\right)$$

The possible quantum states of a single qubit can thus be visualized using a *Bloch sphere* (figure 2.2). The three main axes represent the following states:

- **z-axis** goes from the states $|1\rangle$ (bottom) to $|0\rangle$ (top),
- **x-axis** goes from the states $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ (back) to $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ (front),
- **y-axis** goes from the states $\frac{|0\rangle - i|1\rangle}{\sqrt{2}}$ (left) to $\frac{|0\rangle + i|1\rangle}{\sqrt{2}}$ (right).

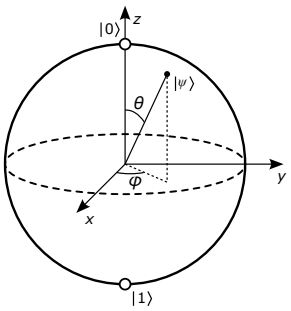



Figure 2.2: The Bloch sphere visualization of the state of a qubit $|\psi\rangle$.

**2** *The Hilbert space of a composite system is the Hilbert space of the tensor product of the state space associated with the component systems.*

This postulate deals with the combination of multiple quantum systems. Taken two qubits $|\psi\rangle$ and $|\phi\rangle$, the composition of the two is represented as $|\psi\rangle \otimes |\phi\rangle$ or $|\psi\rangle|\phi\rangle$ or $|\psi\phi\rangle$ equivalently. This results in a vector of the tensor product of the two state spaces $(\mathbb{C}^2)^{\otimes 2}$ $(= \mathbb{C}^4)$.

$$|\psi\rangle = a|0\rangle + b|1\rangle \qquad |\phi\rangle = c|0\rangle + d|1\rangle$$

$$|\psi\phi\rangle = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

Can be easily seen that the resulting state represents a unit vector in the new state space.

**3** *The time evolution of a closed system is described by a unitary transformation*

$$|\psi'\rangle = U|\psi\rangle$$

In this case, the words *closed system* means that the system described doesn't interchange information (i.e energy and/or matter) with another system. This thesis will only treat closed systems, because a quantum system is extremely sensible to outside interference that can change the state of the qubit or the progress of the computation.

The evolution of the system, or in other words the computation, is therefore described by a unitary matrix $U$. The problem is thus creating an appropriate matrix $U$, implementing the desired operation on the state of the system, that will carry out the computation of the algorithm leading to the solution.

The last postulate deals with the measurement process.

**4** *A projective measurement is described by a hermitian operator $M$ defined as*

$$M = \sum_i \lambda_i P_{\lambda_i}$$

*where $\lambda_i$ are the eigenvalues denoting the possible outcomes of the operation and $P_{\lambda_i} = |\lambda_i\rangle\langle\lambda_i|$ are the projectors matrices for the respective $\lambda_i$. The measurement on a quantum state $|\psi\rangle$ returns the value $\lambda_i$ with a probability calculated as*

$$p(\lambda_i) = \langle\psi|P_{\lambda_i}|\psi\rangle = |\langle\lambda_i|\psi\rangle|^2$$

*After the measurement with result $\lambda_i$ the state becomes*

$$|\psi'\rangle = \frac{P_{\lambda_i}|\psi\rangle}{\sqrt{\langle\psi|P_{\lambda_i}|\psi\rangle}}$$

## 2.5 Circuit based quantum computing

The quantum circuit model [5] is the most used paradigm for the representation of quantum computations. It takes inspiration from its classical counterpart. Graphically the qubits are represented by lines. The computation takes place from left to right. The lines do not necessarily correspond to a physical wire; they may correspond instead to the passage of time, or perhaps through space. At each step, unitary matrices are applied at each qubit progressing the computation. The states of the qubits are thus modified and, at the end, they are measured to get the output result. An example can be seen in Figure 2.3.

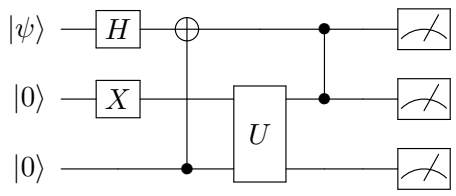



Figure 2.3: Circuit representation example. The initial state of the qubit is visualized on the left of each line. In the first step, a Hadamard gate is applied to the first qubit and an X gate to the second one. It follows a controlled-NOT gate targeting the first qubit with the third one as control. After that, a two qubits gate U is applied to the second and third qubits. Finally, a controlled Z gate is applied to the first two qubits and at the end every qubit is measured.

Several gates have been created that work as building bases for more complex operations. Three gates that act on single qubits are the Pauli gates:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

These base gates are called with specific names: $X$ is the *NOT gate* and $Z$ is called the *$\pi$-phase gate* ($Y$ does not have a special name because it can be derived by the others $Y = iXZ = -iZX$). They also represent rotations of $\pi$ radians around the three main axes. In this regard, there is also a version for the general rotation of an angle $\alpha$ around each axis:

$$R_x(\alpha) = e^{-i\frac{\alpha}{2}X} = \begin{bmatrix} \cos(\frac{\alpha}{2}) & -i\sin(\frac{\alpha}{2}) \\ -i\sin(\frac{\alpha}{2}) & \cos(\frac{\alpha}{2}) \end{bmatrix}$$

$$R_y(\alpha) = e^{-i\frac{\alpha}{2}Y} = \begin{bmatrix} \cos(\frac{\alpha}{2}) & -\sin(\frac{\alpha}{2}) \\ \sin(\frac{\alpha}{2}) & \cos(\frac{\alpha}{2}) \end{bmatrix}$$

$$R_z(\alpha) = e^{-i\frac{\alpha}{2}Z} = \begin{bmatrix} e^{-i\frac{\alpha}{2}} & 0 \\ 0 & e^{i\frac{\alpha}{2}} \end{bmatrix}$$

Other important gates are the *identity gate $I$*, *Hadamard gate $H$* and the *$\alpha$-phase gate $P(\alpha)$*:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad P(\alpha) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

The gate $P(\alpha)$ corresponds to $R_z(\alpha)$, up to a global phase ($e^{-i\frac{\alpha}{2}}$). For some angles, the $\alpha$-phase gate gets different names:

- $\alpha = \pi \rightarrow P(\pi) = Z$,
- $\alpha = \frac{\pi}{2} \rightarrow P(\frac{\pi}{2}) = S$,
- $\alpha = \frac{\pi}{4} \rightarrow P(\frac{\pi}{4}) = T$,
- $\alpha = 0 \rightarrow P(0) = I$.

There is also the necessity of gates acting on more than one qubit. Therefore gates were created affecting two qubits at a time. The most used ones are the *controlled-NOT gate $CNOT$* (or $\wedge X$) and the *controlled-Phase gate $CPHASE$* (or $\wedge Z$):

$$\wedge X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \wedge Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The last of the fundamental gates is a three qubit gate: the *Toffoli gate* (also called the *controlled-controlled-NOT*).

$$Toffoli = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

These gates acting on two or more qubits create a phenomenon called *entanglement*. The qubits are now connected: the state of a single element cannot be fully described without considering the state of the others. This property enables the construction of more complex quantum algorithms allowing the implementation of intricate states connecting multiple qubits.

The combination of these gates allows the implementation of larger unitary matrices. These can act on multiple qubits, evolving the state and implementing a specific computation.

It's been proved that there exist sets of gates that are *universal*: any operation possible on a quantum computer can be expressed as a finite sequence of gates from the set. Some examples of universal quantum gates are the sets:

- $\{R_x(\alpha), R_y(\alpha), R_z(\alpha), P(\alpha), \wedge X\}$ [7],
- $\{\wedge X, H, T\}$ [10],
- $\{\wedge X, R_y(\frac{\pi}{4}), S\}$ [9],
- $\{Toffoli, H\}$ [13],
- $\{\wedge Z, J_\alpha\}$ [15] with $J_\alpha = HP(\alpha)$.

## 2.6 Measurement based quantum computing

A different model for performing quantum computations is now presented. This model is the focus of this thesis. The technique in question takes the name of *Measurement Based Quantum Computing* (*MBQC*), in particular, the version called *One-Way Quantum Computing* (*1WQC*).

This new model was presented in 2001 by Raussendorf, Browne and Briegel [11] and later refined by the same authors in 2003 [14]. It has been proved that the computation power of this new model is equal to the one of the circuit model [18][21][27].

The computation starts from a highly entangled many-qubit state called a *graph state*. This structure is a graph in which the nodes are qubits in the superposition state $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. The edges of this graph represent an entanglement between two qubits. Normally this connection is created by applying a $\wedge Z$ to the pair of qubits. If the graph state is created as a lattice with the connections taken between adjacent qubits, it takes the name of *cluster state*.

Once the graph state has been created, the computation can be performed by measuring the qubits. At each step, a set of qubits is measured moving forward the computation on the connected qubits. These operations are performed by choosing for each qubit the bases in which it is measured. After each step, the connecting qubits get a correction operation depending on the results of the previous measurements. At the end of the computation, in the remaining qubits have been applied a unitary matrix produced by all the measurements done.

In [21] has been proposed a *measurement calculus* that is now reported. The basic components are:

- qubit preparation $N_i$,
- entanglement operator $E_{ij}$,
- measurement $M_i^\alpha$,
- correction operators $X_i$ and $Z_i$.

The indices $i$ and $j$ represent the qubits on which each of these operations apply, and $\alpha$ is an angle in $[0, 2\pi]$ (expressions involving angles are always evaluated modulo $2\pi$). These types of command will be referred to as $N$, $E$, $M$ and $C$, respectively. Let's now describe them in detail.

The preparation operator $N_i$ acts on the qubit $i$ of the graph state. This operation leaves the qubit in the state $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. In other words, if the initial state of the qubit was $|0\rangle$ this operation consists in an application of a Hadamard gate $H$.

The entanglement command $E_{ij}$ acts on qubits $i$ and $j$. This operation consists of the application of a $\wedge Z$ on the two qubits, connecting them.

The measurement command $M_i^\alpha$ acts on the qubit $i$. It's a destructive operation given that the qubit is no longer usable and is no longer part of the graph state. A measurement is defined by the pair of bases used. In this case, a command $M_i^\alpha$ uses as bases $|+_\alpha\rangle = \frac{|0\rangle+e^{i\alpha}|1\rangle}{\sqrt{2}}$ and $|-_\alpha\rangle = \frac{|0\rangle-e^{i\alpha}|1\rangle}{\sqrt{2}}$. The value measured $s_i$ correspond to 0 and 1, respectively. If $\alpha$ takes value 0 the operation corresponds to a measurement on the $X$-axis; if it takes value $\frac{\pi}{2}$ correspond to the one on the $Y$-axis. In this case, a measurement on the $Z$-axis is not taken into account, because this operation removes the qubit from the graph state without changing the state of the neighboring qubits. This particular operation is indeed used to remove qubits from the graph state in order to disconnect pairs of qubits.

Lastly, the correction operators $X_i$ and $Z_i$ act on qubits $i$ applying the corresponding gate. These operations are used to correct the computation according to the result of previous measurements. We write $X_i^{s_j}$ (and $Z_i^{s_j}$) to indicate that the correction operation needs to be applied to the qubit $i$ only if the measurement on qubit $j$ gave as result 1. These dependencies are crucial if one wants to define a universal computing model [21].

A *pattern* consists of three finite sets $(V, I, O)$, with *input* set $I \subseteq V$ and *output* set $O \subseteq V$, and a finite sequence of commands $A_n, ..., A_1$ (read from right to left) applying to qubits $V$ in the order $A_1$ first and $A_n$ last, such that:

1. no command depends on an outcome not yet measured;
2. no command acts on a qubit already measured;
3. no command acts on a qubit not yet prepared;

4. a qubit $i$ is measured if and only if $i$ is not an output, $i \notin O$.

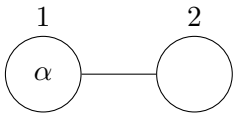
The sequence $A_n, ..., A_1$ of a pattern is called the *command sequence*, while the triple $(V, I, O)$ is called the pattern *type*.

The implementation of a quantum algorithm in 1WQC consists of creating a pattern that satisfies all the above conditions, choosing the proper measurement bases.

Some examples are now presented to aid the comprehension of the model.

### 2.6.1 Examples

Let's start from the simple case of a graph state composed of only two qubits:




In this case, the qubit 1 will be measured with the operation $M_1^\alpha$ and the qubit 2 will be the output. As said before, the bases used for this measurement are $|\pm_\alpha\rangle = \frac{|0\rangle \pm e^{i\alpha}|1\rangle}{\sqrt{2}}$. The branch of the computation that is of interest is the full positive one, i.e. all the measurements gave result 0. In this case, the measurement of the first qubit consists of projecting it into the bases used:

$$\langle +_\alpha|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & e^{-i\alpha} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \equiv 1$$

$$\langle +_\alpha|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & e^{-i\alpha} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \equiv e^{-i\alpha}$$

The normalization factors $\frac{1}{\sqrt{2}}$ have been dropped for notational convenience. So when applied to the initial cluster state $|\Psi\rangle$:

$$|\Psi\rangle = E_{12}\,|{+}{+}\rangle = \tfrac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$
$$\Big\downarrow M_1^\alpha$$
$$\tfrac{1}{2}(|0\rangle + |1\rangle + e^{-i\alpha}\,|0\rangle - e^{-i\alpha}\,|1\rangle) = \tfrac{1}{2} \begin{bmatrix} 1 + e^{-i\alpha} \\ 1 - e^{-i\alpha} \end{bmatrix}$$

It can be easily seen that this state corresponds to $J(-\alpha)\,|+\rangle = HP(-\alpha)\,|+\rangle$. If the result of the measurement would have been 1, i.e. $|-_\alpha\rangle$, the state of the second qubit would be:

$$\frac{1}{2}(|0\rangle + |1\rangle - e^{-i\alpha}\,|0\rangle + e^{-i\alpha}\,|1\rangle) = \frac{1}{2} \begin{bmatrix} 1 - e^{-i\alpha} \\ 1 + e^{-i\alpha} \end{bmatrix}$$

corresponding to $XHP(-\alpha)\,|+\rangle$.

To ensure that the computation is always deterministic, the appropriate adjustments will be made using the corrector operators. Therefore, if the result of the measurement $M_1^\alpha$ of the first qubit is

represented by the binary value $s_1$, the second qubit takes the value:

$$X^{s_1} HP(-\alpha) \left|+\right\rangle$$

It can be easily seen that this result holds also if the initial state of the first qubit is an arbitrary $\left|\psi\right\rangle = a\left|0\right\rangle + b\left|1\right\rangle$. The result in this case would be the transfer of the state to the second qubit with the application of the above-mentioned gates. Graphically:

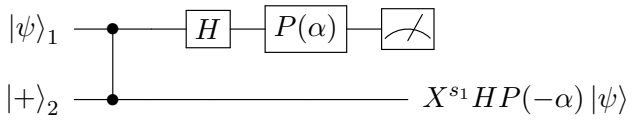
$$E_{12}\left|\psi\right\rangle\left|+\right\rangle = \tfrac{1}{2}\begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix} = \tfrac{1}{2}(a\left|00\right\rangle + a\left|01\right\rangle + b\left|10\right\rangle - b\left|11\right\rangle)$$

$$\downarrow M_1^{\alpha}$$

$$\tfrac{1}{2}(a\left|0\right\rangle + a\left|1\right\rangle + be^{-i\alpha}\left|0\right\rangle - be^{-i\alpha}\left|1\right\rangle) = \tfrac{1}{2}\begin{bmatrix} a + be^{-i\alpha} \\ a - be^{-i\alpha} \end{bmatrix}$$

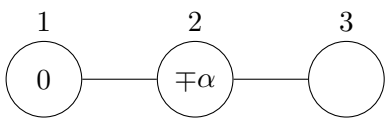
It is easily verifiable that the state of the second qubit is $X^{s_1} HP(-\alpha)\left|\psi\right\rangle$. This result makes the study of 1WQC designs, with a greater number of qubits and thus more complex, easier.

The simple case shown can be translated into a circuit model equivalent. This result has been presented in [18]. The measurement operation can be viewed as a sequence of gates that perform the change of bases followed by a measurement in the standard ones. In this case, the gates convert the bases $\{\left|0\right\rangle, \left|1\right\rangle\}$ to the ones used by the measurement, i.e. $\left|\pm_\alpha\right\rangle$. The 1WQC design is thus implemented as:




Where the change of bases is done by the gates $Z(\alpha)H$ being that $\left|\pm_\alpha\right\rangle = Z(\alpha)H\left|0/1\right\rangle$. The result of the measurement of the first qubit is represented by $s_1$.

Let's take a little bigger example: the design for a rotation on the $x$-axis $R_x(\alpha)$. To create such a gate, the 1WQC design uses the identity $R_x(\alpha) = HR_z(\alpha)H$ (and consequently $R_x(\alpha) = HP(\alpha)H$). The implementation would then be:




This time the first qubit is measured with the bases $\left|\pm\right\rangle$ and the second one with bases $\left|\pm_{\mp\alpha}\right\rangle$. With these operations, the qubit labeled 3 will take the value:

$$X^{s_2} HP(\pm\alpha) X^{s_1} H\left|+\right\rangle$$

The sign of the angle of the measurement of the second qubit is dependent on the result of the first

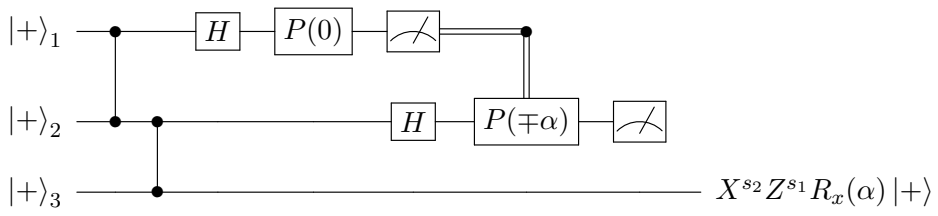
qubit. If the first measurement gives the result 0, then the angle for the qubit 2 will be $-\alpha$. On the other hand, if the first result is 1 the angle becomes $\alpha$. This needs to be done because the $X^{s_1}$ needs to be moved to the left, past the $P(\pm\alpha)$, in order to have all the adjustments as the leftmost and last operations.

The state of the third qubit can be written as:

$$\begin{gathered} X^{s_2} H P(\pm\alpha) X^{s_1} H \left|+\right\rangle \\ \downarrow \\ X^{s_2} H X^{s_1} P(\alpha) H \left|+\right\rangle \\ \downarrow \\ X^{s_2} Z^{s_1} H P(\alpha) H \left|+\right\rangle \\ \downarrow \\ X^{s_2} Z^{s_1} R_x(\alpha) \left|+\right\rangle \end{gathered}$$

The second transformation uses another known identity: $HX = ZH$. This example highlights the structure of the result: at first, the unitary transformation desired is applied, followed by a sequence of $X$ and $Z$ gates based on the results of the previous measurements. These last ones adjust the computation making it always deterministic.

Lastly, let's convert also this design to the circuit model. This will be:



# 3

# Algorithm Description

This section is devoted to the presentation of the algorithms introduced in [17]. The problem solved by the authors is the following: starting from a unitary matrix $U$, create a 1WQC model that implements the matrix. The model will consist of the description of the graph state including the number of nodes and connections between them. Furthermore, for each node, the measurement angles need to be chosen.

The solution is divided into three algorithms: the first one creates a phase map decomposition of the input matrix $U$, the second one chooses the angles of measurements for each qubit and creates the graph state with the connections between the nodes from the phase map created in the previous step, and the last one checks if the generated graph has a causal flow to ensure that the computation of the 1WQC model is correct. For the sake of completeness, the last algorithm will be presented in the version given in [25] since it has a lower time complexity.

## 3.1 Notation

First of all let's introduce the notation that will be used in the algorithms.

The input of this decomposition is a unitary matrix $U$ of size $2^n \times 2^n$, operating thus on $n$ qubits. Therefore the input $I$ is composed of $n$ qubits and the output $O$ is composed of $n$ qubits as well. It can be said w.l.o.g. that the input qubits set and the output one are disjoint: $I \cap O = \emptyset$.

During the computation, the set $W$ of *auxiliary qubits* will be added, where $|W| = m$. It will always be that $m \geq n$ because the set of input and output qubits have the same size and are disjoint. The new qubits added that do not belong to the output are called *pure auxiliary qubits* and belong to the set $A$ with $|A| = a = m-n$. The total set of the qubits used is $V$ and its cardinality is: $|V| = n+a+n = n+m$.

The qubits are indexed with a number from 0 to $|V| - 1$ $(= n + m - 1)$. If at some point there are $k$ qubits and one is inserted, this new qubit takes index $k + 1$. With this numbering, the input qubits will have indices in $\{0, ..., n - 1\}$ and the output ones will have indices in $\{m, ..., n + m - 1\}$.

## 3.2 Idea

A brief explanation of the idea behind the algorithms of [17] is now reported.

Various operators over the Hilbert space $\mathcal{H}$ preserve the computational basis up to a phase (examples

are Pauli gates, $P(\alpha)$ and controlled $X$ and $Z$). The one-qubit measurements also map the standard basis of one space to those of another one up to a scalar factor.

A map $\Phi : \mathcal{H}_V \to \mathcal{H}_V$ is called a *phase map* if it is diagonal in the computational basis and has only unit coefficients. This definition depends on the choice of the bases. A typical example of such a map is the gate $\wedge Z$.

The computation in a 1WQC model can be divided into an initial step where the additional qubits are added to the input ones, followed by the application of a phase map and, in the end, the restriction of the qubits to the output ones as a result of the measurements.

To visualize the addition of qubits in the $|+\rangle$ state, is defined the *preparation map* $P_{I\to V} : \mathcal{H}_I \to \mathcal{H}_V$ that expands the input space by tensoring auxiliary qubits:

$$|x\rangle \to |x\rangle \otimes |+\cdots+\rangle_{I^c}$$

This operation introduces a constant factor of $\frac{1}{\sqrt{2^m}}$.

The operation of restricting the state space is defined as a *restriction map* $R_{V\to O} : \mathcal{H}_V \to \mathcal{H}_O$ that projects the space to the output one:

$$|x\rangle \to \langle+\cdots+|_{O^c}|x\rangle$$

Measurement and entangling commands in the one-way model define phase maps. Hence, from the universality of the model, the following decomposition is obtained:

**Theorem.** *For all unitary $U : \mathcal{H}_I \to \mathcal{H}_O$, there exists a phase map $\Phi : \mathcal{H}_V \to \mathcal{H}_V$ such that:*

$$U = R_{V\to O} \circ \Phi \circ P_{I\to V}$$

*Proof.* The proof is presented in [17] (Chapter 3) and follows from the universality of $\{\wedge Z, J_\alpha\}$ ($J_\alpha = HP(\alpha)$) [15] and compositions of flows [16]. □

This decomposition can be seen as a special kind of diagonalization for unitaries where one is allowed to inflate the dimension of the underlying space. The reader may notice that this decomposition is not physical, since the phase map does not directly correspond to any operation.

To determine whether a phase map decomposition $R\Phi P$ of a unitary $U$ has a corresponding pattern in the one-way measurement model, one needs to find a graph $G(V, E)$ and the angles $\alpha_j$ for each $j \in O^c$ such that:

$$\Phi = \prod_{j\in O^c} P(-\alpha_j) \prod_{jk\in E} \wedge Z_{jk}$$

This means that for all $z \in \{0, ..., 2^{|V|} - 1\}$:

$$d_{zz} = e^{-1\sum_{j\in O^c} \alpha_j z_j} (-1)^{\sum_{jk\in E} z_j z_k}$$

Where $z_j$ represents the $j$-th bit of the binary representation of $z$ and $d_{zz}$ is the diagonal coefficient of the phase map $\Phi$ corresponding to the $z$ row/column.

From the graph created, the corresponding geometry $(G, I, O)$ can be derived using the first $n$ qubit as the input set $I$ and the last $n$ qubit for the output set $O$. After choosing the measurement angles and creating the connections between qubits, all that remains is to verify that the geometry created has a *causal* flow. This needs to be done to create the dependencies between qubits measurements and to verify that the are no cycles in them. This is needed to have a deterministic computation [16]. The dependencies defined by the flow of the geometry exposes how information flows within it during the execution of a pattern.

The following example, extended from [17], can be useful to make the reader more familiar with the aforementioned notion.

### 3.2.1 Example

Consider the pattern $X_3^{s_2} Z_3^{s_1} M_2^0 M_1^{-\alpha} E_{12} E_{23}$ which implements the $P(\alpha)$ gate. $V = \{1, 2, 3\}$, $I = \{1\}$ and $O = \{3\}$. Hence, $n = 3$ and $m = 2$. The positive branch, i.e. when the measurements results are all 0, is:

$$\langle +|_2 \langle +_{-\alpha}|_1 E_{12} E23| + ++\rangle$$

which induces the 3-qubit phase map $\Phi|xyz\rangle = (-1)^{xy+yz} e^{i\alpha y}|xyz\rangle$. This corresponds to the following decomposition of $P(\alpha)$:

$$R_{12} D(1, 1, 1, -1, e^{i\alpha}, e^{i\alpha}, -e^{i\alpha}, e^{i\alpha}) P_{23}$$

where $D(...)$ is a diagonal matrix representing $\Phi$, $P_{23} : \mathcal{H}_{\{1\}} \to \mathcal{H}_{\{123\}}$ and $R_{12} : \mathcal{H}_{\{123\}} \to \mathcal{H}_{\{3\}}$.

The matrices $R_{12}$, $\Phi$ and $P_{23}$ are as follows:

$$U = R_{12} \cdot \Phi \cdot P_{23} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{i\alpha} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\alpha} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -e^{i\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\alpha} \end{bmatrix} \cdot \frac{1}{\sqrt{2^2}} \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

It can be seen that the preparation matrix $P$ and the restriction matrix $R$ have opposite dimensions. However, they are not one the conjugate transpose of the other (contrary to what was said in [17]) because they act by adding and removing different qubits. This clarification does not affect the validity of the decomposition.

## 3.3 Algorithms

The overall computation is divided into three parts:

1. The first one creates a phase map decomposition of the input matrix $U$ acting on $n$ qubits. It returns the diagonal matrix $\Phi$ represented as a vector. The elements represent the values of the

main diagonal of $\Phi$.

2. The second one finds, if they exist, the connections between qubits from the phase map created in the previous step. This process creates the graph state and defines the angle of measurement for each qubit not in the output set.

3. The last part is dedicated to finding the flow of the geometry created. If the geometry has no flow, false is returned. On the other hand, if the flow is present: the function $f$, representing the dependency relation, and the labelling $L$ are returned. This last one indicates the set of nodes that can be measured at the same time and in which order these sets need to be measured.

Reusing the example presented in 3.2.1, the algorithms described correspond to the following operations.

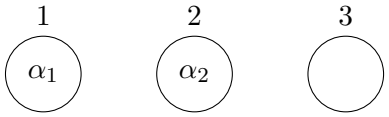
1. The input matrix $U$ is:

$$U = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

The first algorithm takes this matrix and creates the phase map decomposition $\Phi$ represented by the array $D$ (composed of the elements of its main diagonal):

$$D = \begin{bmatrix} 1 & 1 & 1 & -1 & e^{i\alpha} & e^{i\alpha} & -e^{i\alpha} & e^{i\alpha} \end{bmatrix}$$

2. The second algorithm creates the graph state with $n + m$ nodes, in this case, 3 nodes.




After that for each node in $O^c$, it sets the measurement angle. So:

$$100_\text{b} = 4 \;\rightarrow\; d[4] = e^{-i\alpha_1} \;\rightarrow\; e^{i\alpha} = e^{-i\alpha_1} \;\rightarrow\; \alpha_1 = -\alpha$$

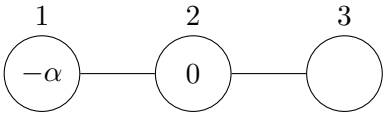
$$010_\text{b} = 2 \;\rightarrow\; d[2] = e^{-i\alpha_2} \;\rightarrow\; 1 = e^{-i\alpha_2} \;\rightarrow\; \alpha_2 = 0$$

The nodes of the output are taken as if they have a measurement angle of 0. With these angles set, the algorithm checks for each pair of nodes $jk$ if they are consistent. Doing that it also finds the connections in the graph. So for the nodes of this example:

- $jk = 12 \;\rightarrow\; 110_\text{b} = 6 \;\rightarrow\; d[6] = \pm e^{-i(\alpha_1+\alpha_2)} \;\rightarrow\; -e^{i\alpha} = \pm e^{-i(-\alpha+0)} \;\rightarrow\; e_{12} \in E$
- $jk = 13 \;\rightarrow\; 101_\text{b} = 5 \;\rightarrow\; d[5] = \pm e^{-i(\alpha_1+\alpha_3)} \;\rightarrow\; e^{i\alpha} = \pm e^{-i(-\alpha+0)} \;\rightarrow\; e_{13} \notin E$
- $jk = 23 \;\rightarrow\; 011_\text{b} = 3 \;\rightarrow\; d[3] = \pm e^{-i(\alpha_2+\alpha_3)} \;\rightarrow\; -1 = \pm e^{-i(0+0)} \;\rightarrow\; e_{23} \in E$

So the first and third pair of nodes give origin to an edge. On the other hand, the second pair doesn't create an edge but respects the equality with the positive sign. Therefore the graph state

is:




3. The third algorithm takes the graph state created in the previous step and checks for a causal flow. The output will be the elements $(f, L)$, where $L$ represents the partial order $\preccurlyeq$. In this case the algorithm returns:

$$f(1) = 2 \quad f(2) = 3$$

$$L = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix} \quad \rightarrow \quad \text{qubit } 1 \preccurlyeq \text{qubit } 2 \preccurlyeq \text{qubit } 3$$

So the graph state can be represented as the one in Figure 3.1.

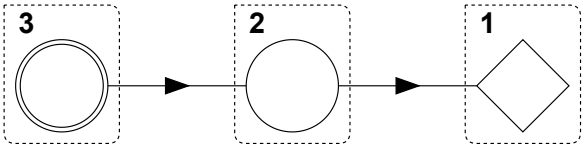



Figure 3.1: The double-circled node is the input qubit and the diamond is the output one. The flow map $f$ is represented by the arrows and the labelling of the 3 partition sets correspond to the value of $L$.

### 3.3.1 Algorithm 1 - phase map

The first algorithm 1 takes in input the unitary matrix $U$ of $2^n \times 2^n$ elements and creates a possible phase map decomposition.

**Algorithm 1** Phase map decomposition

1: **function** PHASE DECOMPOSITION$(n, U)$
2: choose $m > 2n$
3: create $x_{pq}^{(j)}$ for each $j \in \{0, ..., 2^a - 1\}$ and $p, q \in \{0, ..., 2^n - 1\}$
4: **for all** $p, q \in \{0, ..., 2^n - 1\}$ **do**
5: choose $x_{pq}^{(j)}$ respecting the constraints: $u_{pq} = \sum_{j=0}^{2^a-1} x_{pq}^{(j)}$ and $\sqrt{2^m}\ |x_{pq}^{(j)}| = 1$
6: **end for**
7: $\sigma \leftarrow$ permutation of $\{0, ..., 2^a - 1\}$
8: $d \leftarrow$ array of length $2^{n+m}$
9: **for** $k \in \{0, ..., 2^{n+m} - 1\}$ **do**
10: $p \leftarrow$ restriction of $k$ to bits of $O$
11: $q \leftarrow$ restriction of $k$ to bits of $I$
12: $j \leftarrow \sigma(\lfloor \frac{k}{2^n} \rfloor)$
13: $d_k \leftarrow \sqrt{2^m}\ x_{pq}^{(j)}$
14: **end for**
15: **return** $d$
16: **end function**

Note that the variables $x_{pq}^{(j)}$ need to be multiplied by a constant factor of $\sqrt{2^m}$ (introduced by the preparation map) when the unit value is imposed (line 5 last equation) and when they are translated to phase map elements (line 13).

This algorithm creates one possible decomposition of the unitary matrix. Running it multiple times gives multiple decompositions that can be tested in the next steps. Another way of thinking is that if a possible decomposition fails the tests of the next steps, one can come back to create a new different possible phase map decomposition. All these arrays will be tested to verify that they can be implemented because the decompositions created by this algorithm do not take into consideration the structure of the 1WQC model.

Let's now focus on the core of the algorithm (line 5). It consists in choosing the values of each $x_{pq}^{(j)}$ respecting the two rules:

$$u_{pq} = \sum_{0 \leq j \leq 2^a - 1} x_{pq}^{(j)}$$

$$\sqrt{2^m}\ |x_{pq}^{(j)}| = 1$$

These values can be taken using different methods (they will be discussed in the next chapter 4.4.1).

On lines 10 and 11 the values of $p$ and $q$ are created restricting the binary representation of $k$ to the bits associated with the qubits in $I$ and $O$, respectively. If the indices of the qubits are taken as shown in section 3.1, these two lines can be rewritten as:

$$p \leftarrow k \mod 2^n$$
$$q \leftarrow k \text{ rshift } m$$

Where the operation "rshift $m$" consists of shifting a binary number $m$ position to the right. Practically, $p$ gets the value of the $n$ rightmost bits of $k$ and $q$ the value of the $n$ leftmost bits of $k$ (because $|k| = n+m$).

### 3.3.2 Algorithm 2 - graph state

The second algorithm 2 takes as input a phase map decomposition $\Phi$ and the number of qubits $n + m = |V|$. It finds, if possible, the graph associated with such decomposition.

This algorithm sets the value of each angle of measurement using the corresponding values of the phase map (line 7). After that, it checks if the combinations of such angles are consistent with the angles associated with each pair of qubits (line 11). If such a connection can be created it will be added to the graph. If, on the other hand, the corresponding element in the phase map is inconsistent, then a graph for this decomposition cannot exist. In this last case, the phase map is incorrect and therefore it is necessary to go back to the first algorithm 1 to create a new decomposition.

### 3.3.3 Algorithm 3 - causal flow

The last part of the algorithm consists of finding a causal flow 3. Is now reported the algorithm proposed in [25].

The function $N(v)$ with $v \in V$ represents the neighborhood of the node $v$ (i.e. $N(v) = \{w \mid vw \in E\}$).

This algorithm recursively finds the nodes to which a set of nodes is dependent, starting from the output nodes $O$.

At each step, the set of nodes $C$ represents a set of potential nodes that can possibly depend on the ones in $V \setminus \texttt{Out}$. The nodes in $C$ are called *correctors* because they can correct the indeterminate result of the measurement of the ones in `Out` making the computation deterministic. The set $C'$ contains the

**Algorithm 2** Graph creation

**function** GRAPH CHECKER($n+m$, $d$)
  $V \leftarrow$ set of $n+m$ nodes
  $G \leftarrow (V, E = \emptyset)$
  $\alpha \leftarrow$ array of length $n+a$
  **for** $j \in \{0, ..., n+a-1\}$ **do**
    $z \leftarrow |V|$-bit string with a 1 only at position $j$ (left to right)
    $\alpha_j \leftarrow$ value such that $e^{-i\alpha_j} = d_{zz}$
  **end for**
  **for all** $j, k \in \{0, ..., |V|-1\}$ **do**
    $z \leftarrow |V|$-bit string with a 1 only at positions $j$ and $k$ (left to right)
    **if** $d_{zz} == \pm e^{-i(\alpha_j+\alpha_k)}$ **then**
      **if** the sign is $-1$ **then**
        $E \leftarrow E \cup \{e_{jk}\}$
      **end if**
    **else**
      **return** $(\emptyset, \emptyset)$ ▷ there is no matching graph
    **end if**
  **end for**
  **return** $G$
**end function**

**Algorithm 3** Creation of the causal flow from the geometry.

**function** FLOW FINDER($G$, $I$, $O$)
  $f \leftarrow$ map from $O^c$ to $I^c$
  $L \leftarrow$ array of $|O|$ elements
  **for all** $v \in O$ **do**
    $L(v) \leftarrow 0$
  **end for**
  **return** FLOW AUX($G$, $f$, $L$, $I$, $O$, $O$, 1)
**end function**
**function** FLOW AUX($G$, $f$, $L$, $I$, $\mathtt{Out}$, $C$, $k$)
  $\mathtt{Out}' \leftarrow \emptyset$
  $C' \leftarrow \emptyset$
  **for all** $v \in C$ **do**
    **if** $\exists u$ s.t. $N(v) \cap (V \setminus \mathtt{Out}) == \{u\}$ **then**
      $f(u) \leftarrow v$
      $L(v) \leftarrow k$
      $\mathtt{Out}' \leftarrow \mathtt{Out}' \cup \{u\}$
      $C' \leftarrow C' \cup \{v\}$
    **end if**
  **end for**
  **if** $\mathtt{Out}' == \emptyset$ **then**
    **if** $\mathtt{Out} == V$ **then**
      **return** (true, $(f, L)$) ▷ a flow has been found and returned
    **else**
      **return** (false, $(f, [\,])$) ▷ a flow is not possible
    **end if**
  **else**
    FLOW AUX($G$, $f$, $L$, $I$, $\mathtt{Out} \cup \mathtt{Out}'$, $(C \setminus C') \cup (\mathtt{Out}' \cap (V \setminus I))$, $k+1$)
  **end if**
**end function**

nodes that are actually used as correctors. The set $\texttt{Out}'$ represents the nodes on which the nodes of $C'$ depend, thus corrected by the latter.

The flow returned is defined by: the dependency map $f$ and the partial order represented by the labelling $L$. Each node/qubit is associated with a *layer* given by $L$. The elements of the same layer can be measured in parallel being that there is no dependence between them. The last layer to be measured is the layer 1, while the first one is the one with the biggest value. The proof of the correctness of this algorithm is presented in detail in [25].

In [25] it is also proven that the flow generated by Algorithm 3 has minimal depth, i.e. the number of layers associated with the qubits is minimal. Therefore this value poses an upper bound on the depth of the corresponding deterministic one-way quantum computation.

This algorithm replaces the third one proposed in [17]. This has been done because the algorithm reported 3 has a quadratic complexity with respect to the number of nodes in the graph (as will be demonstrated in 4.3). Instead of the cubic complexity, with respect to the number of nodes in the graph, that the third algorithm in [17] has.

# 4

# Complexity Analysis

The complexity analysis of the algorithms reported in the previous section will be presented in this chapter. For each part, possible implementations will be proposed to ease the computation. The resulting complexity will depend on the techniques used for the implementation of the three parts.

## 4.1 Algorithm 1 - phase map

Algorithm 1 takes as input a unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$. With this in mind, the complexity will be calculated with respect to an input of size $2^{2n}$.

Let's analyze each step of the algorithm. The first task is to choose the value of $m$: the number of qubits to be added ($I^c$). As proven in [17] this value needs to be greater than two times the number of the input qubits (in this case $n$), i.e. $m \geq 2n$, to allow the presence of a phase map. This is a constant operation with time and space complexity of $\Theta(1)$ and $\Theta(\log n)$, respectively.

The second line creates the variables $x_{pq}^{(j)}$ that will compose the phase map. These elements need to be complex numbers. The total number of these variables is $2^{n+a+n}$ being that there are $2^a$ "slots" over the diagonal for each element of the input matrix $U$. The subscripts $p$ and $q$ indicate the row and column respectively of the element of $U$ associated, and the superscript $(j)$ represents the index of the "slot". For a better implementation, these variables can be taken as elements of the form

$$x_{pq}^{(j)} = \pm \frac{1}{\sqrt{2^m}} \, e^{i\alpha_{pq}^{(j)}}$$

This technique allows to shift the problem from the choice of $x_{pq}^{(j)}$ to the selection of the overall sign and the value of the angle $\alpha_{pq}^{(j)}$. The new variables $\alpha_{pq}^{(j)}$ are constrained between the values $[-\frac{\pi}{2}, +\frac{\pi}{2}]$. A slightly different way can be used if one wants to eliminate the problem of choosing the sign: the variables $\alpha_{pq}^{(j)}$ can be taken in the range $[-\pi, \pi]$. This is possible because $e^{i\alpha} = -e^{i(\alpha+\pi)}$, taking the resulting angle $\mod 2\pi$.

The resulting complexity of line 3 is $\Theta(2^{n+a+n}) = \Theta(2^{n+m})$ in time and space.

After the creation of the variables, the next step consists in assigning their value. In line 5 each $x_{pq}^{(j)}$ is set with respect to two equations:

$$u_{pq} = \sum_{0 \leq j \leq 2^a - 1} x_{pq}^{(j)}$$

$$\sqrt{2^m}\ |x_{pq}^{(j)}| = 1$$

The first one imposes that the restriction map $R$ sums up all the $x_{pq}^{(j)}$ over $j$ to give the value of $u_{pq}$. The second equation enforces the variables to be of unit value. If the technique presented for their implementation is used, this last rule is always satisfied because $|e^{i\alpha}| = 1$ for every value of $\alpha$. The constant $\sqrt{2^m}$ is needed to balance the one introduced by the preparation map $P$ which consists of the constant $\frac{1}{\sqrt{2^m}}$.

This operation has a space complexity of $\Theta(2^{n+m})$ because of the space occupied by the variables. The time complexity of assigning the values is $\Theta(2^{n+a+n}) = \Theta(2^{n+m})$. The techniques used for choosing the values are discussed in 4.4.1.

After the first for loop, two new variables are created. On line 7 the array $\sigma$ gets a permutation of the elements in $\{0, ..., 2^a - 1\}$. This operation takes $\Theta(2^a)$ both in time and space complexity. On line 8 the last variable $d$ is created as an array of $2^{n+m}$ elements. The complexity in this case is the same as the previous line, that is $\Theta(2^n)$ both in time and space.

The last operation consists of a for loop of $2^{n+m}$ cycles. At each iteration, four assignments are executed. Line 10 and 11 can be executed in $n$ steps each using the labelling of the qubits presented in section 3.1, having thus a time and space complexity of $\Theta(n)$. The creation of $j$ is done by accessing the array $\sigma$, an operation feasible in $\Theta(1)$ time. And the last is the assignment of the element of $d$, also doable in $\Theta(1)$. The for loop therefore has an overall time complexity of $\Theta(n \cdot 2^{n+m}) \equiv \Theta(2^{n+m})$ and space complexity of $\Theta(n + m)$ (because of the index).

The number of qubits added $m$ (and therefore also $a$) can be treated as linear with respect to $n$, i.e. $m = kn$ with $k \in \mathbb{Z}$ and $k$ does not depend on $n$.

> *The function* `PHASE DECOMPOSITION` *of the algorithm* 1 *has an overall complexity of* $\Theta(2^n)$ *in time and* $\Theta(2^n)$ *in space.*

The algorithm is therefore linear in time and space with respect to the input unitary matrix $U$.

## 4.2 Algorithm 2 - graph state

Algorithm 2 creates the graph state. It takes in input the number of qubits $n+m = |V|$ and the diagonal matrix $\Phi$, represented by the array $d$ of $2^{n+m}$ elements. It returns a graph with $V$ as the nodes and $E$ as the set of the new edges created. The complexity is therefore calculated with respect to the input array of length $2^{n+m}$ of complex numbers.

The first three lines (2,3 and 4) create the variables used in the algorithm. The first two create a graph $G$ with the set $V$ as the nodes and an empty set $E$ for the edges. The graph can be implemented as an ordered adjacency list: an array of $|V|$ elements where each entry is a list of nodes (indices) that represents the neighbors of the node. The elements in these lists will be maintained in ascending order of indices. Every element of these lists will maintain the reference to the next and previous element. With this structure, the insertion of a new edge will have a complexity of $O(\log(n + m))$ in time using a binary search to find the exact spot in the lists of the two indices involved. Therefore the creation of the graph with an empty set of edges has a complexity of $\Theta(n + m)$ in time and space.

Line 4 creates an array of $n + a = m$ angles, which takes $\Theta(m)$ in space and time complexity.

After that follows a for loop over each element of the array $\alpha$ just created. At each iteration a $|V|$-bit string of zeros with only a 1 is constructed. This operation has a time and space complexity of $\Theta(n+m)$. Inside the loop, there is also the assignment of the element of $\alpha$ with a simple operation. If the technique presented in 4.1 is used, this step consists only of the change of sign of the angle. This line has thus a complexity of $\Theta(1)$ in time and space. The for loop has therefore an overall complexity of $\Theta(m(n+m))$ in time and $\Theta(n+m)$ in space.

The next step of the algorithm consists of another for loop, this time over all the pairs $jk$ with $j$ and $k$ elements of the set $\{0, ..., |V|\}$. Inside the loop, at line 10, a $|V|$-bit string of zeros with 1 only in the positions of $j$ and $k$ is created. Like in the previous for loop this operation has a complexity of $\Theta(n+m)$ in time and space. This operation is followed by two if statements one inside the other. The checks of the two conditions are simple and feasible in constant time. The operation of line 13 consists of adding a new edge to the set $E$. With the implementation presented earlier, this action can be done with a time complexity of $O(\log(n+m))$ and space complexity of $\Theta(\log(n+m))$ for the indices of the nodes. Thus, the for loop has an overall complexity of $\Theta\left((n+m)^2(n+m)\right) = \Theta\left((n+m)^3\right)$ in time and $\Theta(n+m)$ in space.

> *The function* `GRAPH CHECKER` *of the algorithm* 2 *has an overall complexity of* $\Theta((n+m)^3)$ *in time and* $\Theta(n+m)$ *in space.*

The algorithm is therefore logarithmic in time and space with respect to the length of the input array $d$, which is $2^{n+m}$, and polynomial in time and linear in space with respect to the number of nodes $|V|$.

## 4.3 Algorithm 3 - causal flow

Algorithm 3 consists of the search and creation of a causal flow (if present). The function `FLOW FINDER` takes in input a graph $G(V, E)$ ($|V| = n+m$), created at the previous step, and two sets of nodes $I$ and $O$ which are subsets of the set of all the nodes $V$. The set of edges $E$ normally would contain a number of edges in the $O(|V|^2) = O((n+m)^2)$. It has been demonstrated [24] that in a geometry with a causal flow, the number of edges $E$ is less than $k|V| - \binom{k+1}{2}$ where $k$ is the size of the output set, so in this case $|E| \leq n(n+m) - \binom{n+1}{2}$. This result can be used to check whether the input graph respects this constraint, and if not, reject the input as the flow cannot exist.

The first lines are dedicated to the creation and initialization of the variables associated with the flow structure. On line 2 the function $f$ is created. This function takes as input a qubit and returns another qubit, where this last one is dependent on the measurement of the first one. This variable can be implemented as an array of $m$ elements where at each index is stored the value of the returned qubit. The creation of this structure has a complexity of $\Theta(m)$ both in time and space. On line 3 the creation of the array $L$ of $m$ elements follows. It represents the labelling for the order of execution of the measurement of each qubit. This can be viewed as the partial order of the flow, only that it is reversed: the qubits $j$ with $L(j) = 1$ will be measured last. The operations of creating the labelling and initializing it all to 0 have a $\Theta(m)$ time and space complexity.

After the creation of the variables of the flow, the procedure is relegated to an auxiliary function called `FLOW AUX`.

This second function takes in input: the graph $G$, the variables representing the flow, the input and output sets, a set of possible initial correctors and the value associated with the first layer, i.e. 1.

The second-last set of nodes, called in the function $C$ for *correctors*, contains the nodes that can be used to correct the computation on the one-way model using the result of a previous measurement. This set is, therefore, initialized with the set of output nodes. The set of nodes already corrected is called `Out` and is initialized with the output set $O$. At each step, this set will be incremented by $\texttt{Out}'$ which represents the set of nodes that are corrected by the correctors in $C'$, which will be removed from $C$.

As reported in [25], for each node $v$, the algorithm can be decomposed in the following steps:

- insert $v$ in the set of potential correctors $C$,
- create the set of vertices that $v$ might correct $N(v) \cap (V \setminus \texttt{Out})$ with the set `Out` of that instant,
- check if $v$ can be used to correct some vertex, i.e. $|N(v) \cap (V \setminus \texttt{Out})| = 1$,
- when $v$ is removed, update the potential correctors sets $N(u) \cap (V \setminus \texttt{Out})$ of the remaining nodes $u \in C$.

The complexity analysis that is proposed takes a slightly different route from the one presented in [25] but comes to the same result. In the paper cited, the author uses a different structure to represent the sets $N(v) \cap (V \setminus \texttt{Out})$ which depends on the number of edges. Hence, the complexity depends on the value of $|E|$, later limited by the result presented in [24]. This allows to represent the complexity based only on the number of nodes. In this thesis, the complexity result will depend directly on the number of nodes of the graph.

Taken a node $v$, the structure used for the set $N(v) \cap (V \setminus \texttt{Out})$ will be implemented as an array of length $|V|$ with each entry representing a node of $V$: it takes the value 1 if it is in the set, 0 otherwise. The algorithm keeps track of the number of 1 in such a structure. This counter will be used to check if $|N(v) \cap (V \setminus \texttt{Out})| == 1$ in constant time. So the creation of the structure for the node $v$ will have a time and space complexity of $\Theta(|V|)$.

These arrays will always be one for each node in $C$. So, at each recursive call, the memory will contain a matrix $S$ of $|C| \times |V|$ elements. Therefore, this matrix will change in size during the computation.

When the node $v$ is removed from the set $C$, the updating of the structures of the remaining nodes in $C$ must be done. This update consists of the removal of the entry associated with $v$ from their entries in $S$. This operation can be done in constant time for each node, so in total it has a time complexity of $\Theta(|C|)$. While updating the structures, it can be checked if the modified array has only one element different from 0 (using the associated counter). If it is true, the corresponding node can be saved to ease the execution of the for loop and the if checks, located at line 12 and 13, respectively.

For each node the function has an overall time complexity of $\Theta(|V| + |C|)$ and space one of $\Theta(|V|)$. The set $C$ contains nodes, so it can't be greater than $V$. Therefore the overall complexity of the auxiliary function will be $O(|V|^2)$ both in time and space (being that at each recursive call the number of structures maintained is $|C|$).

`FLOW FINDER` *has an overall complexity of* $O((n+m)^2)$ *in time and* $O((n+m)^2)$ *in space.*

This function is therefore polynomial (quadratic to be precise) in time and space with respect to the number of nodes of the input graph $G$.

## 4.4 Overall complexity

The algorithm for the phase map decomposition is thus composed of 3 parts: phase map creation 1, measurement angles assignment and graph state creation 2 and causal flow search 3.
To recap:

**Alg 1** input: unitary matrix $U$ of $2^n \times 2^n$ numbers $\longrightarrow$ complexity: $\Theta(2^n)$ time and $\Theta(2^n)$ space,

**Alg 2** input: number $n+m$ and array $d$ of length $2^{n+m}$ $\longrightarrow$ complexity: $\Theta((n+m)^3)$ time and $\Theta(n+m)$ space,

**Alg 3** input: graph $G$ of $n+m$ nodes $\longrightarrow$ complexity: $O((n+m)^2)$ time and $O((n+m)^2)$ space.

Hence, the overall complexity of the procedure is as follows.

> *The algorithm for phase decomposition of a unitary matrix $U$ of size $2^n \times 2^n$ has a complexity of $\Theta(2^n)$ in time and $\Theta(2^n)$ in space.*

The overall procedure is therefore linear in time and space with respect to the input unitary matrix $U$. This result means that each time a value for the variables $x_{pq}^{(j)}$ is chosen, the procedure takes $\Theta(2^n)$ in both time and space to check if it is a possible translation into a 1WQC implementation. If one of the steps fail, i.e. there is no possible graph state or it does not have a causal flow, there are multiple ways to backtrack and try with a new decomposition:

- select a different permutation $\sigma$ (Algorithm 1 line 7)
- select new values for the variables $x_{pq}^{(j)}$ (Algorithm 1 line 5)
- change the number of qubits added $m$ (Algorithm 1 line 2)

### 4.4.1 Variables assignment

The only task not yet analyzed is how to choose the values for the variables $x_{pq}^{(j)}$.

In [17], the authors proposed a technique to create a possible assignment. Starting form a random initial assignment the steps are:

- create the set $L = \emptyset$ of locked elements
- choose a pair $(x_{p_1q_1}^{(h)}, x_{p_2q_2}^{(k)})$ not in $L$ with $p_1, q_1, p_2, q_2 \in \{0, ..., 2^n - 1\}$ and $h, k \in \{0, ..., 2^a - 1\}$
- take the angle associated with them $(\alpha_{p_1q_1}^{(h)}, \alpha_{p_2q_2}^{(k)})$
- choose an angle $\theta \in [0, \frac{\pi}{2}]$
- rotate the chosen angles by $\theta$ in the opposite way: $\alpha_{p_1q_1}^{(h)} \leftarrow \alpha_{p_1q_1}^{(h)} + \theta$ and $\alpha_{p_2q_2}^{(k)} \leftarrow \alpha_{p_2q_2}^{(k)} - \theta$
- check if the sum of the variables is equal to $U$: $u_{pq} = \sum_{j=0}^{2^a-1} x_{pq}^{(j)}$ for each $pq$ with $p, q \in \{0, ..., 2^n-1\}$
- if, for some $pq$, the equation returns true then lock the associated variables: $L \cup \{x_{pq}^{(j)} | j \in \{0, ..., 2^a-1\}\}$

- if all the elements are locked, i.e. $|L| = 2^{2n+a}$, exit the procedure, otherwise return to the first step

This procedure will iterate until a possible decomposition of $U$ has been found. The rotations $\theta$ can also be restricted to be multiples of $\frac{\pi}{4}$ to help with the implementation [20].

Each valid assignment of the $x_{pq}^{(j)}$ variables can be viewed as a solution in a space of all the possible choices. Therefore, below is proposed a possible implementation where the problem of finding a valid assignment of the variables $x_{pq}^{(j)}$ can be viewed as a *local search* problem. A local search algorithm [8] consists of a procedure that finds possible solutions moving in the search space using operations that modify a solution to give a new one. These operations are called *moves*. The procedure uses specific heuristics, based on the type of algorithm used and the problem analyzed, to evaluate each possible solution and move in the search space. It returns a possible solution when a specific termination criterion has been reached.

An implementation in this regard would need to specify the following steps:

- *initialization*,
- *evaluation*,
- *neighbors generation*,
- *selection*,
- *termination*.

The initialization step of the variables $x_{pq}^{(j)}$ could be done in multiple ways. Two examples are: generating random values for all of them without concern about the sum, or finding for each pair $pq$, with $p, q \in \{0, ..., 2^n - 1\}$, a possible solution to the equations:

$$u_{pq} = x_{pq}^{(0)} + x_{pq}^{(1)} + ... + x_{pq}^{(2^a-1)}$$

Remember that with the implementation of the variables presented in 4.1 the second type of equation (i.e. all variables need to be of unit value) can be ignored.

The next step consists of the evaluation of a given solution to determine how "good" it is. To do this, an appropriate function must be used. In this case, a possible candidate could be the *Mean Squared Error* (MSE) between the calculated $u_{pq}$, given by the sum of the current values of the variables $x_{pq}^{(j)}$, and the corresponding entry of the matrix $U$. If this error is equal to zero, the solution generated can be used as an exact phase map decomposition and the optimization problem could terminate returning the found values. The solutions that have an MSE different from zero could still be useful. This variable assignment represents an approximation of the implementation of the unitary matrix $U$, and the value of the error would be associated with the precision of such an approximated result. So in a case where the optimization problem is taking too much time to find an exact solution, these approximated ones can be used for the phase map.

To find new possible solutions, the local search algorithm needs to create the neighbors of the current one making a move. This operation can be done using the technique reported previously (presented in

[17]): a pair of variables is chosen and rotated by the same amount in the opposite way $\pm\theta$. The set of neighbors is thus dependent on the choice of which variables to modify and the value of the rotation $\theta$.

The selection step consists of choosing which of the neighbors would be the next temporary solution and making the associated move. The criterion to use for the choice could be finding the best MSE of these new solutions. This value can be calculated in a smart way: modifying the value of the calculated $u_{pq}$ corresponding to the sets of only the two variables modified, having thus to do $2^{a+1}$ (or $2^a$ in the case that the two variables belong to the same $u_{pq}$) steps and not $2^{2n+a}$.

The final step consists in specifying the criteria used to terminate the search algorithm, i.e. when to stop the search for a better solution. These can be: the solution has MSE $= 0$, reached the maximum number of iterations or maximum time of computation.

When a solution is found, Algorithms 1, 2 and 3 test if it is a valid phase decomposition for $U$. If the tests fail, the search for a new solution can restart from this one, applying a move to a worse neighbor and imposing to not come back to the previous solution.

This local search problem could thus be implemented using the heuristics of *simulated annealing* together with elements of *tabu search*. The first one allows to reach better and more diverse results, traversing different parts of the search space even if they are worse with respect to the evaluation function. The second one instead implements a tabu table necessary to avoid returning to an already tested solution.

# 5

# Conclusion and future works

The One Way Quantum Computing model has sparked interest since its conception in [11]. This interest is due to its intuitively simpler realization. To this day some possible realizations could be promising [32] but they are not yet implemented and open to the public like for the circuit model. Nevertheless, the research on this topic has never stopped.

One of the most promising paths is the translation of circuit-based algorithms (described as unitaries) into the one-way setting. Usually, the technique used to achieve this translation consists of decomposing the unitary matrix in simpler gates and then implementing them in the 1WQC model using the known transformations presented in [14]. This method could however introduce redundant operations and unnecessary qubits in the graph state.

The scope of this thesis was to inspect and analyze the algorithm proposed in [17] for the implementation of a unitary matrix directly in a 1WQC model. A complexity analysis has been done on all the parts of the algorithm, with the last one replaced by the better and faster version proposed in [25]. The technique studied here could open the way to better realization of complex algorithms using fewer qubits, finding possible implementation based each time on the matrix analyzed. Further work is needed to exploit the full potential of such a model.

Future works could consist of the implementation of these algorithms using the local search structure presented. The implementation could be tested on different unitary matrices, for example the ones created for the Quantum Random Walk problem presented in [12][36].

The implementation as a local search problem could be extended adding the constraint imposed by the second algorithm on the existence of the graph associated with a possible phase map. These equations could be integrated with the ones used for the generation of a possible solution and used in the evaluation function as a further error in the case that they are not respected.

# Bibliography


[1] Roman S. Ingarden. Quantum information theory. *Reports on Mathematical Physics*, 10(1):43–72, 1976.

[2] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22:563–591, 05 1980.

[3] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6/7):467–488, 1982.

[4] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400:117 – 97, 1985.

[5] David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425:73 – 90, 1989.

[6] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439:553 – 558, 1992.

[7] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467, Nov 1995.

[8] Marc Pirlot. General local search methods. *European Journal of Operational Research*, 92(3):493–511, 1996.

[9] A. Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, December 1997.

[10] P. Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing, 1999.

[11] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, May 2001.

[12] Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. Quantum walks on graphs, 2002.

[13] Yaoyun Shi. Both toffoli and controlled-not need little help to do universal quantum computation, 2002.

[14] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2), August 2003.

[15] Vincent Danos, Elham Kashefi, and Prakash Panangaden. Parsimonious and robust realizations of unitary maps in the one-way model. *Physical Review A*, 72(6), December 2005.

[16] Vincent Danos and Elham Kashefi. Determinism in the one-way model. *Physical Review A*, 74(5), November 2006.

[17] Niel de Beaudrap, Vincent Danos, and Elham Kashefi. Phase map decompositions for unitaries, 2006.

[18] Michael A. Nielsen. Cluster-state quantum computation. *Reports on Mathematical Physics*, 57(1):147–161, February 2006.

[19] Daniel E Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9(8):250–250, August 2007.

[20] Vincent Danos and Elham Kashefi. Pauli measurements are universal. *Electronic Notes in Theoretical Computer Science*, 170:95–100, 2007. Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005).

[21] Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus, 2007.

[22] Niel de Beaudrap. A complete algorithm to find flows in the one-way measurement model, 2007.

[23] Niel de Beaudrap. Finding flows in the one-way measurement model. *Physical Review A*, 77(2), February 2008.

[24] Niel de Beaudrap and Martin Pei. An extremal result for geometries in the one-way measurement model, 2008.

[25] Mehdi Mhalla and Simon Perdrix. *Finding Optimal Flows Efficiently*, page 857–868. Springer Berlin Heidelberg, 2008.

[26] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[27] Einar Pius. Automatic parallelisation of quantum circuits using the measurement based quantum computing model. 2010.

[28] Agung Trisetyarso and Rodney Van Meter. Circuit design for a measurement-based quantum carry-lookahead adder. *International Journal of Quantum Information*, 08(05):843–867, August 2010.

[29] P. M. Alsing, A. M. Smith, M. L. Fanto, C. C. Tison, and G. E. Lott. Programming non-trivial algorithms in the measurement-based quantum computation model. In Keith L. Lewis, Richard C. Hollins, Thomas J. Merlet, Alexander Toet, Mark T. Gruneisen, Miloslav Dusek, and John G. Rarity, editors, *Emerging Technologies in Security and Defence II; and Quantum-Physics-based Information Security III*, volume 9254, page 92540I. International Society for Optics and Photonics, SPIE, 2014.

[30] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*, 4:279, June 2020.

[31] Michele Mosca and Priyanka Mukhopadhyay. A polynomial time and space heuristic algorithm for t-count. *Quantum Science and Technology*, 7(1):015003, oct 2021.

[32] Swapnil Nitin Shah. Realizations of measurement based quantum computing, 2021.

[33] Péter Rakyta and Zoltán Zimborás. Approaching the theoretical limit in quantum gate decomposition. *Quantum*, 6:710, May 2022.

[34] Péter Rakyta and Zoltán Zimborás. Efficient quantum gate decomposition via adaptive circuit compression, 2022.

[35] Carla Piazza, Riccardo Romanello, and Robert Wille. An asp approach for the synthesis of cnot minimal quantum circuits. In *Italian Conference on Computational Logic*, 2023.

[36] D. Della Giustina, C. Londero, C. Piazza, B. Riccardi, and R. Romanello. Quantum encoding of dynamic directed graphs. *Journal of Logical and Algebraic Methods in Programming*, 136:100925, 2024.